MATH 3027 Optimization 2021: Coursework 1

Mingdong He

Question 1

Find all of the stationary points of f:

$$f(x,y) = 2y^4 - x^2 + 1 + (x^2 + 2y^2 - 1)^2.$$

Calculate the gradient of function f:

$$\nabla f = \begin{pmatrix} (-2x + 2(x^2 + 2y^2 - 1)(2x) \\ 8y^3 + 2(x^2 + 2y^2 - 1)(4y)) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Simply two equations, we can get

$$x(2x^2 + 4y^2 - 3) = 0, y(x^2 + 3y^2 - 1) = 0$$

Solve them we can get the stationary points:

$(0,0), (0, -\frac{1}{\sqrt{3}}), (0, \frac{1}{\sqrt{3}}), (-\frac{\sqrt{6}}{2}, 0), (\frac{\sqrt{6}}{2}, 0)$

Compute the Hessian:

$$\nabla^2 f(x,y) = \begin{pmatrix} 12x^2 + 8y^2 - 6 & 16xy \\ 16xy & 8x^2 + 72y^2 - 8 \end{pmatrix}$$

Evaluate stationary points:

$\nabla^2 f(0,0) = \begin{pmatrix} -6 & 0 \\ 0 & -8 \end{pmatrix}$, $\nabla^2 f(0, -\frac{1}{\sqrt{3}}) = \begin{pmatrix} -\frac{10}{3} & 0 \\ 0 & 16 \end{pmatrix}$, $\nabla^2 f(0, \frac{1}{\sqrt{3}}) = \begin{pmatrix} \frac{10}{3} & 0 \\ 0 & 16 \end{pmatrix}$, $\nabla^2 f(-\frac{\sqrt{6}}{2}, 0) = \begin{pmatrix} 12 & 0 \\ 0 & 4 \end{pmatrix}$, $\nabla^2 f(\frac{\sqrt{6}}{2}, 0) = \begin{pmatrix} 12 & 0 \\ 0 & 4 \end{pmatrix}$,

Since all non-zero elements appear on the diagnal, it is obvious that they are the eigenvalues of each matrix.

1. $(0,0)$ is a strict local maximum, since Hessian matrix is negative definite.
2. $(-\frac{\sqrt{6}}{2}, 0), (\frac{\sqrt{6}}{2}, 0)$ are strict local minimum, since Hessian matrices are positive definite.
3. $(0, -\frac{1}{\sqrt{3}}), (0, \frac{1}{\sqrt{3}})$ are saddle points, since their Hessians have both positive and negative eigenvalues so undefinite.

Question 2

```
load(file="R/CW1_PimaData.rda")
head(X)
```

```
##   pregnant glucose pressure
## 1        6     148       72
## 2        1      85       66
## 3        8     183       64
## 4        1      89       66
## 5        0     137       40
## 6        5     116       74
```

```r
head(y)
```

```
##   diabetes
## 1        1
## 2        0
## 3        1
## 4        0
## 5        1
## 6        0
```

```r
fit<-glm(diabetes~.-1,family=binomial,data=Pima.dat)
coef(fit)
```

```
##     pregnant      glucose     pressure
##  0.056509905  0.008079294 -0.022841519
```

```r
h<-function(z){
  1/(1+exp(-z))
}
X_pos=X[y==1,]
X_neg=X[y==0,]

loglike<-function(theta){
  sum(log(h(X_pos%*%theta)))+sum(log(1-h(X_neg%*%theta)))
}
```

i.Gradient:

Formula derivation:

Intuition: By observation, we found that:

$$h(t)' = (\frac{1}{1+e^{-t}})' = \frac{e^{-t}}{(1+e^{-t})^2} = \frac{1}{1+e^{-t}}(1-\frac{1}{1+e^{-t}}) = h(t)(1-h(t))$$

Similarly, considering the coefficient in front of h:

$$h(x_i^T\theta)' = h(x_i^T\theta)(1-h(x_i^T\theta))x_i$$

Thus:

$$
\begin{aligned}
\nabla l(\theta) &= \nabla(\sum_{i=1}^{n} y_i \log h(x_i^T\theta) + (1-y)\log(1-h(x_i^T\theta))) \\
&= \sum_{i=1}^{n}(\frac{y_i h(x_i^T\theta)(1-h(x_i^T\theta))x_i}{h(x_i^T\theta)} - (1-y_i)\frac{h(x_i^T\theta)(1-h(x_i^T\theta))x_i}{1-h(x_i^T\theta)}) \\
&= \sum_{i=1}^{n}(y_i(1-h(x_i^T\theta))x_i - (1-y_i)h(x_i^T\theta)x_i) \\
&= \sum_{i=1}^{n}(y_i - h(x_i^T\theta))x_i \\
&= X^T(y-\hat{y}(\theta))
\end{aligned}
$$

as desired if we use matrix form followed by question.

```
gradient_l<-function(theta){
  y_hat<-h(X%*%theta)
  out<-t(X)%*%(y-y_hat)
  return(out)
}
# input is 3 by 1 vector
theta<-matrix(0,nr=3)
gradient_l(theta)
```

```
##          diabetes
## pregnant    -24.5
## glucose    -816.0
## pressure   -753.5
```

Numerical checking by finite difference approximation:

```
theta<-matrix(0,nr=3)
eps<-10^-5
central_diff<-function(theta,eps){
  d1<-(loglike(theta+c(eps,0,0))-loglike(theta-c(eps,0,0)))/(2*eps)
  d2<-(loglike(theta+c(0,eps,0))-loglike(theta-c(0,eps,0)))/(2*eps)
  d3<-(loglike(theta+c(0,0,eps))-loglike(theta-c(0,0,eps)))/(2*eps)
  return(c(d1,d2,d3))
}
central_diff(theta,eps)
```

```
## [1]  -24.5 -816.0 -753.5
```

The result is consistent with previous one.

ii. Gradient decent method:

```
#Max l(theta) ~ Min -l(theta)
gradient_Desent<-function(theta,t_bar,Print){
  trajectory<-theta
  for (i in 2:10^6){
    theta<-theta+t_bar*gradient_l(theta)
    grad=-gradient_l(theta)

    if (sqrt(sum(grad^2)) < 10^-3){
      break
```

```r
    }
    trajectory<-cbind(trajectory,theta)
  }
  if(Print){
      fun_val=loglike(theta)  # we don't need this unless we are printing it out
      print(paste('-------------- Iteration ',i-1, ' -------------'))
      print(paste("l(theta) = ", signif(fun_val,3), " norm_grad = ", signif(sqrt(sum(grad^2)),3)))
  }
  return(trajectory)
}

theta<-matrix(0,nr=3)
t_bar=10^-6
trajectory<-gradient_Desent(theta,t_bar,Print = TRUE)
```

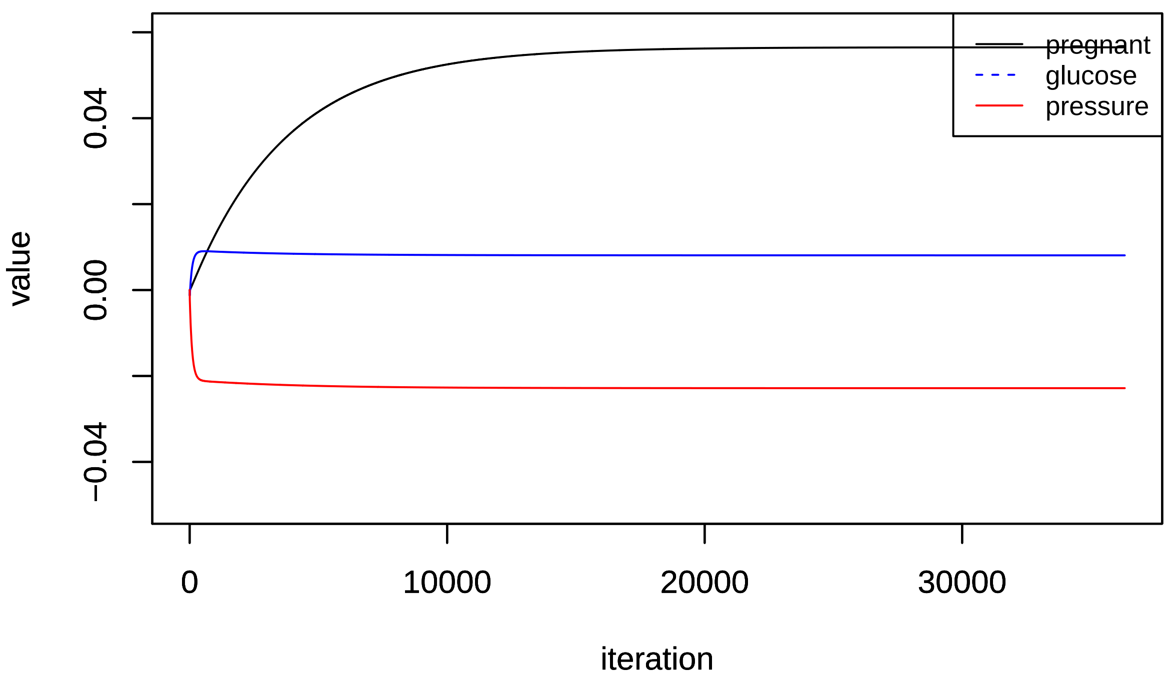
```
## [1] "-------------- Iteration  36315  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.001"
```

Optimal value is:

```
##     pregnant      glucose     pressure
##  0.056506126  0.008079365 -0.022841382
```

Plot the trajectory:

```r
plot(trajectory[1,],type="l",col='black',xlab='iteration',ylab='value',ylim=c(-0.05,0.06))
par(new=TRUE)
plot(trajectory[2,],type="l",col='blue',xlab='iteration',ylab='value',ylim=c(-0.05,0.06))
par(new=TRUE)
plot(trajectory[3,],type="l",col='red',xlab='iteration',ylab='value',ylim=c(-0.05,0.06))
legend(x = "topright",legend=c("pregnant", "glucose",'pressure'),col=c("black", "blue","red"), lty=1:2,
```



iii: larger stepsize

using $\bar{t} = 10^{-5}$

In fact, this program can not converge if we use $\bar{t} = 10^{-5}$. The reason might be that the stepsize is too large, e.g:oscillation might happens so it cannot reach the optimal point.

iv. Backtracking:

```
gradient_Desent_Backtracking<-function(theta,Print,s,alpha,beta){
  stopifnot(beta>0,beta<1,alpha>0,alpha<1)
  trajectory<-theta
  for (i in 1:10^6){
    fun_val<-loglike(theta)
    grad=gradient_l(theta)
    t_bar<-s
    while (loglike(theta+t_bar*grad) - fun_val< (alpha*t_bar*sum(grad^2))){
      t_bar<-beta*t_bar
    }
    theta<-theta+t_bar*gradient_l(theta)
    if (sqrt(sum(grad^2)) < 10^-3){
      break
    }
    trajectory<-cbind(trajectory,theta)

  }

  if(Print){
      grad<-gradient_l(theta)
      fun_val<-loglike(theta)  # we don't need this unless we are printing it out
      print(paste('-------------- Iteration ', i-1, ' -------------'))
      print(paste("l(theta) = ", signif(fun_val,3), " norm_grad = ", signif(sqrt(sum(grad^2)),3)))
  }
  return(trajectory)
}
```

Run our code to get the iterations and optimal

```
theta<-matrix(0,nr=3)
z<-gradient_Desent_Backtracking(theta,Print = TRUE,1,0.25,0.5)
```

```
## [1] "-------------- Iteration  7835  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.00158"
```

Optimal value of θ is

```
##    pregnant     glucose    pressure
##  0.05650686  0.00807935 -0.02284141
```

It works much better than the gradient descent method.

Try other parameters:

If we change α,

$\alpha = 0.5$

```
## [1] "-------------- Iteration  2815  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.00271"
```

$\alpha = 0.6$

```
## [1] "-------------- Iteration  2524  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.00315"
```

$\alpha = 0.8$

```
## [1] "-------------- Iteration  1930  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.00396"
```

$\alpha = 0.9$

```
## [1] "-------------- Iteration  2833  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.0019"
```

when $\alpha = 0.8$ controlling other parameters, our program works well.

If we change β

$\beta = 0.2$

```
## [1] "-------------- Iteration  2082  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.0035"
```

$\beta = 0.3$

```
## [1] "-------------- Iteration  9373  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.00155"
```

when $\beta = 0.2$ controlling other parameters, our program works well.

Discussion:

1. Changing parameters will change the efficiency of our code.
2. Small change could make a huge difference, sometimes our code is sensitive to them.
3. There do exist a better parameters, like $s = 1, \alpha = 0.8, \beta = 0.5$, which only requires less than 2000 iterations. So we need to do more research of how to improve the efficiency of our code.

v. Computing the Hessian matrix:

Since we have already known that:

$$\begin{aligned}\nabla l(\theta) &= \sum_{i=1}^{n}(y_i - h(x_i^T\theta)x_i) \\ &= X^T(y - \hat{y}(\theta))\end{aligned}$$

Let's take one more derivative to get the Hessian:

$$\begin{aligned}H = \nabla^2 l(\theta) &= \frac{\partial}{\partial\theta^T}(\sum_{i=1}^{n}(y_i - h(x_i^T\theta)x_i) \\ &= \sum_{i=1}^{n} x_i h(x_i^T\theta)(h(x_i^T\theta) - 1)x_i^T \\ &= X^T A X\end{aligned}$$

where $A^{n\times n}$ is a diagonal matrix:

$$A = \begin{pmatrix} h(x_1^T\theta)(h(x_1^T\theta)-1) & ... & 0 \\ . & ... & . \\ 0 & .. & h(x_n^T\theta)(h(x_n^T\theta-1) \end{pmatrix}$$

```r
Hessian<-function(theta){
  A<-matrix(0,nr=100,nc=100)
  for (i in 1:100){
    for (j in 1:100){
      if(i==j){
         A[i,j]<-h(t(X[i,]%*%theta))*(h(t(X[i,]%*%theta))-1)
      }

    }
  }
  out<-t(X)%*%A%*%X
  return(out)
}
theta<-matrix(0,nr=3)
Hessian(theta)
```

```
##             pregnant    glucose   pressure
## pregnant    -850.25  -14343.25   -8262.25
## glucose   -14343.25 -375875.50 -205490.75
## pressure   -8262.25 -205490.75 -127827.25
```

Check the result numerically:

```r
library(numDeriv)
numDeriv::hessian(loglike,theta)
```

```
##              [,1]       [,2]       [,3]
## [1,]   -850.2502  -14343.25   -8262.25
## [2,] -14343.2500 -375875.50 -205490.75
## [3,]  -8262.2499 -205490.75 -127827.25
```

They are consistent!

vi. Pure Newton's method:

```r
pure_Newton <- function(theta,Print){
  trajectory<-theta
  for (i in 2:10^6){
    grad<-gradient_l(theta)
    theta<-theta-solve(Hessian(theta),grad)
    if (sqrt(sum(grad^2)) < 10^-3){
      break
      }
    trajectory<-cbind(trajectory,theta)

    if(Print){
     fun_val<-loglike(trajectory[,i])
     print(paste('-------------- Iteration ', i-1, ' -------------'))
     print(paste("l(theta) = ", signif(fun_val,3), " norm_grad = ", signif(sqrt(sum(grad^2)),3)))
   }
  }
```

```
  return(trajectory)

}
```

Run our code to get the iterations and optimal value.

```
theta<-matrix(0,nr=3)
pure_Newton(theta,Print=TRUE)
```

```
## [1] "-------------- Iteration  1  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  1110"
## [1] "-------------- Iteration  2  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  41.9"
## [1] "-------------- Iteration  3  -------------"
## [1] "l(theta) =  -64.9  norm_grad =  0.387"
```

```
##                diabetes     diabetes     diabetes
## pregnant 0  0.05143148  0.056437600  0.056509891
## glucose  0  0.00748262  0.008071119  0.008079293
## pressure 0 -0.02124781 -0.022820389 -0.022841515
```

After 3 iterations, the algorithm converges, which is significantly faster. The maximum likelihood estimate of θ is

```
##     pregnant      glucose     pressure
##  0.056509891  0.008079293 -0.022841515
```

vii.Regularized

we modify our code for regularized function:

```
#target function to be minimized
l_regular<-function(theta,lambda){
  -loglike(theta)+lambda*(sum(theta^2))
}
#gradient of target function
grad_regular<-function(theta,lambda){
  -gradient_l(theta)+2*lambda*theta
}

#Hessian matrix of target function
Hessian_regular<-function(theta,lambda){
  -Hessian(theta)+2*lambda*diag(c(1,1,1))
}

#Newton's method applied to regularized function
pure_Newton_regular <- function(theta,lambda,Print){
  trajectory<-theta
  for (i in 2:10^6){
    grad<-grad_regular(theta,lambda)
    theta<-theta-solve(Hessian_regular(theta,lambda),grad)

    if (sqrt(sum(grad^2)) < 10^-3){
      break
    }
    trajectory<-cbind(trajectory,theta)
    if(Print){
```

```r
    fun_val<-loglike(theta)
    print(paste('-------------- Iteration ', i-1, ' -------------'))
    print(paste("l(theta) = ", signif(fun_val,3), " norm_grad = ",signif(sqrt(sum(grad^2)),3)))
   }
  }
  return(trajectory[,i-1]) #return the optimal value of theta

}
```

Run our code and get the iterations and optimal value of θ.

```r
theta<-matrix(0,nr=3)
Hessian_regular(theta,10^3)
```

```
##           pregnant   glucose  pressure
## pregnant  2850.25  14343.25   8262.25
## glucose  14343.25 377875.50 205490.75
## pressure  8262.25 205490.75 129827.25
```

```r
pure_Newton_regular(theta,10^3,Print=TRUE)
```

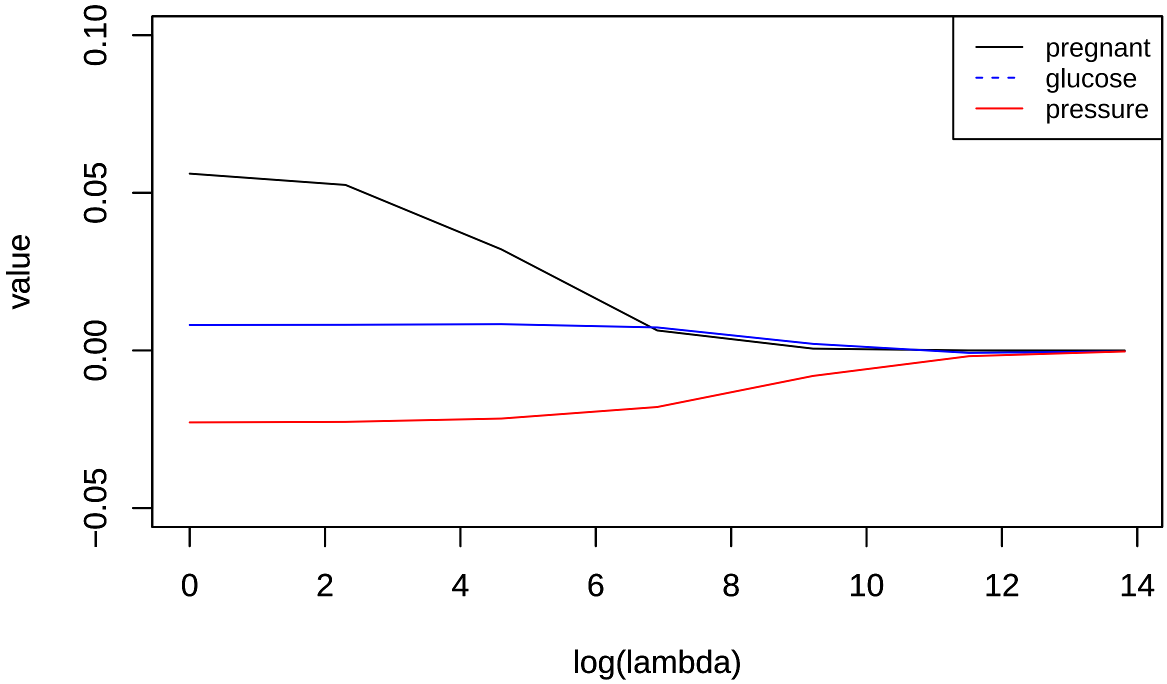
```
## [1] "-------------- Iteration  1  -------------"
## [1] "l(theta) =  -65.3  norm_grad =  1110"
## [1] "-------------- Iteration  2  -------------"
## [1] "l(theta) =  -65.3  norm_grad =  30.8"
## [1] "-------------- Iteration  3  -------------"
## [1] "l(theta) =  -65.3  norm_grad =  0.0934"

##     pregnant      glucose     pressure
##  0.006343552  0.007297859 -0.017947490
```

Plot the estimated value of parameters as λ varies from 1 to 10^6, using a log-scale λ.

```r
lambda<-c(1,10,100,10^3,10^4,10^5,10^6)
estimate<-matrix(0,nr=3,nc=length(lambda))
for (i in 1:7){
  estimate[,i]<-pure_Newton_regular(theta,lambda[i],Print=FALSE)

}
# log-scale for lambda
plot(log(lambda),estimate[1,],type="l",col='black',xlab='log(lambda)',ylab='value',ylim=c(-0.05,0.1))
par(new=TRUE)
plot(log(lambda),estimate[2,],type="l",col='blue',xlab='log(lambda)',ylab='value',ylim=c(-0.05,0.1))
par(new=TRUE)
plot(log(lambda),estimate[3,],type="l",col='red',xlab='log(lambda)',ylab='value',ylim=c(-0.05,0.1))
legend(x = "topright",legend=c("pregnant", "glucose",'pressure'),col=c("black", "blue","red"), lty=1:2,
```



Question 3

Data initialization

```
t<-matrix(seq(0.25,2,0.25),nr=8)
Y<-matrix(c(19.956,17.528,15.987,14.445,9.631,6.663,2.134,0.121),nr=8)
```

i. Coding for functions

function of $\theta = (g, k)^T$:

```
#function f(t,g,k)
f<-function(theta,t){
  g<-theta[1]
  k<-theta[2]
  20-(g/k)*(t+exp(-k*t)/k-1/k)
}

#function s(g,k)
s_gk<-function(theta){
  sum((f(theta,t)-Y)^2)
}

#gradient of function f(t,g,k)
gradient_f<-function(theta,t){
  g<-theta[1]
  k<-theta[2]
  d1<--1/k*(1/k*exp(-k*t)-1/k+t)
  d2<-g*exp(-k*t)/k^3*(k*t+exp(k*t)*(k*t-2)+2)
  return(matrix(c(d1,d2),nr=2))
}
```

```r
#gradient of function s(g,k)
gradient_s_gk<-function(theta,t){
  sg=0
  sk=0
  for (i in 1:8){
    sg<-sg+2*(f(theta,t[i])-Y[i])*(gradient_f(theta,t[i])[1])
    sk<-sk+2*(f(theta,t[i])-Y[i])*(gradient_f(theta,t[i])[2])
  }
  return (c(sg,sk))
}
```

Calculate the gradient:

```r
theta<-c(10,2)
gradient_s_gk(theta,t)
```

```
## [1] -49.31635 158.32410
```

Check our answer numerically, we can see that they are consistent.

```r
numDeriv::grad(s_gk,theta)
```

```
## [1] -49.31635 158.32410
```

ii. Contour plot for $g \in [1, 20], k \in [0, 5]$:

Note that we use $k \in [0.5, 5]$ as a replacement to avoid NaN.

```r
library(pracma)
```

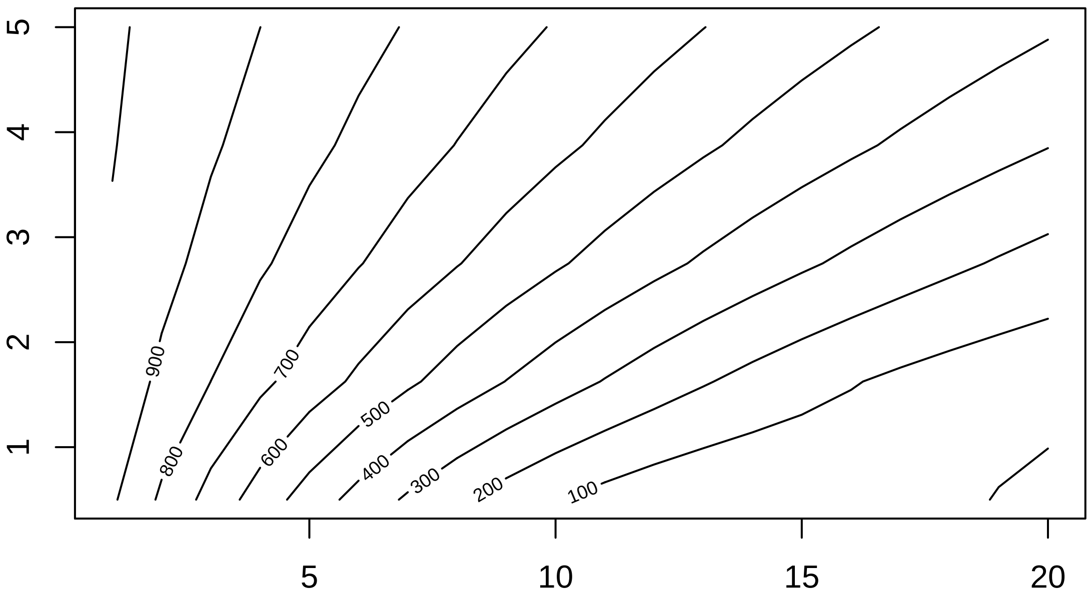
```
##
## Attaching package: 'pracma'

## The following objects are masked from 'package:numDeriv':
##
##     grad, hessian, jacobian
```

```r
#initialization
g<-seq(1,20,length.out=20)
k<-seq(0.5,5,length.out=5)
out<-matrix(0,nr=length(g),nc=length(k))
for (i in 1:length(g)){
  for (j in 1:length(k)){
    out[i,j]<-s_gk(c(g[i],k[j]))
  }
}
```

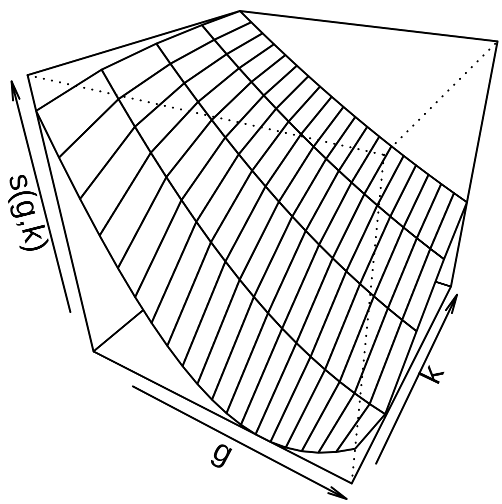
Let's get the contour plot.

```r
#visualize above value by contour plot
contour(g,k,out)
```



We could also have a clear picture of what is going on about our function by its whole perspectives.

```
#whole view
persp(g,k,out,zlab='s(g,k)',theta =30,phi=30)
```



iii. Gauss-Newton algorithm

```
Gauss_Newton<-function(theta,t,Y,Print){
  J<-matrix(0,nr=8,nc=2)
  F<-matrix(0,nr=8,nc=1)
  trajectory<-theta
  for (i in 2:10^3){
    #Calculate the J F
    for (j in 1:8){
    J[j,]<-t(gradient_f(theta,t[j]))
    F[j,]<-f(theta,t[j])-Y[j]
    }

    #Calculate gradient
```

```
    grad<-2*t(J)%*%F
    theta<-theta-0.5*solve(t(J)%*%J,grad)

    #Stopping criteria
    if (sqrt(sum(grad^2)) < 10^-3){
      break
      }

    trajectory<-cbind(trajectory,theta)

    #Print
    if(Print){
     fun_val<-sum((f(theta,t)-Y)^2)
     print(paste('-------------- Iteration ', i-1, ' -------------'))
     print(paste("s(g,k) = ", signif(fun_val,3),"norm_grad = ",signif(sqrt(sum(grad^2)),3)))
     print(paste("g =",signif(trajectory[1,i]),"k =",signif(trajectory[2,i])))
     }
  }
}
```

Run our code and get the iterations and optimal value of θ.

```
#starting from (g,k) = (10,2)
theta<-matrix(c(10,2),nr=2)
Gauss_Newton(theta,t,Y,Print=TRUE)
```

```
## [1] "-------------- Iteration  1  -------------"
## [1] "s(g,k) =  7360 norm_grad =  166"
## [1] "g = 15.9431 k = -1.26274"
## [1] "-------------- Iteration  2  -------------"
## [1] "s(g,k) =  553 norm_grad =  17900"
## [1] "g = 11.0531 k = -0.781458"
## [1] "-------------- Iteration  3  -------------"
## [1] "s(g,k) =  45.9 norm_grad =  1980"
## [1] "g = 13.5184 k = 0.0600452"
## [1] "-------------- Iteration  4  -------------"
## [1] "s(g,k) =  5.05 norm_grad =  282"
## [1] "g = 17.2024 k = 0.803489"
## [1] "-------------- Iteration  5  -------------"
## [1] "s(g,k) =  3.91 norm_grad =  29.8"
## [1] "g = 18.839 k = 1.07368"
## [1] "-------------- Iteration  6  -------------"
## [1] "s(g,k) =  3.9 norm_grad =  0.88"
## [1] "g = 18.8604 k = 1.07874"
## [1] "-------------- Iteration  7  -------------"
## [1] "s(g,k) =  3.9 norm_grad =  0.00168"
## [1] "g = 18.8574 k = 1.07835"
```

After 7 iterations, my program converges to optimal value $(g, k) = (18.8574, 1.07835)$ with the value of objective as $s(g, k) = 3.9$.